Conditional Generative Adversarial Network
A Reimplementation Attempt for ECE-50024 Final Project

Suhaib Khan, MEM, Graduate [1]

Abstract

In recent years, Generative Adversarial Networks, also abbreviated as GANs have seen major adoption in fields like Image Generation, Image-to-Image Translation, Text-to-Image Translation, etc. GANs propose a unique Adversarial Learning structure where two networks, commonly known as Generator and Discriminator work in tandem to train each other without additional supervision. However, the unsupervised nature of vanilla GANs restrict their usage in fields which require the output to be supervised. To alleviate this shortcoming of vanilla GANs, Conditional Generative Adversarial Networks, also abbreviated as cGANs were introduced. cGANs provide an extension on vanilla GANs which allows them to take input and generate an output corresponding to that specific input. This paper is a reimplementation attempt of the original Conditional Generative Adversarial Networks paper by Mehdi Mirza and Simon Osindero.

1. Introduction

Generative Adversarial Network, or GAN is a type of Deep Generative Network that has achieved tremendous success in the recent years. GANs have numerous applications in fields like Natural Language Processing, Image Translation, Computer Vision, Text-to-Image Translation, etc, and are known for their high-quality data generation that closely mimics real data.

Unlike other Deep Generative Networks like Variational Autoencoders (VAE), GANs were introduced by (Goodfellow et al., 2014) and utilized a novel concept of Adversarial Learning. This new discovery allows GANs to avoid probability computations in Maximum Likelihood Estimation that are seemingly intractable. In GANs, two separate networks – Discriminator and Generator are involved in an adversarial minimax game. The Generator takes a latent vector as an input and attempts to produce an image mimicking a real image, while the Discriminator tries to distinguish between the real and synthetic image. This can be easily understood as a competition between a counterfeiter (Generator) who tries to create fake currency notes, and a detective (Discriminator) who tries to distinguish between real and fake currency. The detective then effectively "teaches" the counterfeiter to generate better quality fake currency, and in turn works on itself to distinguish the improved fake currency from the real currency. This task is achieved through backpropagation of error between the Generator and the Discriminator and repeats until convergence.

Even though GANs where widely accepted, the original version suffered from a major drawback. The drawback being the inability to generate consistent output pertaining to a specific input. Since vanilla GAN was completely unsupervised, there was no way of predicting what the generated output would look like. Coming back to the previous example of a counterfeiter and detective, the counterfeiter had no control on whether it generates fake US Dollars, or Indian Rupees. (Mirza & Osindero, 2014) introduced a supervised solution to vanilla GANs which allowed the Generator to take an input label and generate a corresponding output image. These GANs were called Conditional Generative Adversarial Networks or cGANs.

This paper attempts to reimplement the original paper "Conditional Generative Adversarial Networks" by (Mirza & Osindero, 2014) on CIFAR10 dataset using the original DCGAN structure for Generator and Discriminator with the loss function being Wasserstein Loss with Gradient Penalty.

The rest of the paper is structured as follows:

1. Section 2: Literature Review.
2. Section 3: Method.
3. Section 4: Experiment and Results.
4. Section 5: Conclusion.
5. Section 6: GitHub Repository.

[*]Equal contribution [1]Masters in Engineering Management, Purdue University, West Lafayette, USA. Correspondence to: Cieua Vvvvv <c.vvvvv@googol.com>, Eee Pppp <ep@eden.co.uk>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

2. Literature Review

2.1. Generative Adversarial Networks

2.1.1. Concept

Generative Adversarial Networks by (Goodfellow et al., 2014) was the original paper that paved the way for GANs. The main objective in GANs is to learn the Generator's probability distribution p_g over data x. For that, the authors defined a latent vector z which is derived from prior p_z. Then the Generator $G(z; \theta_g)$ was defined as a Multi-Layer Perceptron mapping z from latent space to data space having parameters θ_g. The authors then defined the Discriminator Multi-Layer Perceptron as $D(x; \theta_d)$ that takes in the input data and returns a scalar representing the probability of whether the data was real or it came from the generator p_g. Then both the networks are simultaneously trained such that D can maximize the probability of assigning the correct label to true data and synthetic data, and G can minimize $log(1 - D(G(z)))$. This leads to the Value Function $V(G, D)$ and the cost function is the two-player minimax game between G and D.

$$\begin{aligned} \min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] \\ +E_{z\sim p_z(z)}[log(1 - D(G(z)))] \end{aligned} \tag{1}$$

2.1.2. Algorithm

For a vanilla GAN, the algorithm is defined in Algorithm 1.

Algorithm 1 Training of Generative Adversarial Networks using Stochastic Gradient Descent

Initialize θ_g and θ_d
for number of epochs **do**
  Get minibatch of latent vectors $[z^1 \dots z^M]$
  Get minibatch of true data $[x^1 \dots x^M]$
  Get Discriminator Loss Gradient

$$\frac{1}{M}\nabla_{\theta_d}\sum_{i=1}^{M}[[logD(x^i)] + [log(1 - D(G(z^i)))]] \tag{2}$$

  Get Generator Loss Gradient

$$\frac{1}{M}\nabla_{\theta_g}\sum_{i=1}^{M} log(1 - D(G(z^i))) \tag{3}$$

  Update θ_g and θ_d according to Stochastic Gradient Descent
end for

2.1.3. Advantages / Disadvantages

The most evident advantage of GANs over previous methods was the generative capabilities. Also, the use of Adversarial Learning allows GANs to avoid intractable probability calculation in Maximum Likelihood Estimation and completely eliminate the use of Markov Chains.

The disadvantage of using vanilla GANs is that using Multi-Layer Perceptron for Generator and Discriminator significantly increases the number of learnable parameters, making the training extremely slow. Also, using the Cross-Entropy loss requires careful Generator and Discriminator construction and Hyperparameter tuning and even a slight change may cause the network to produce gibberish output. Additionally, the unsupervised nature of vanilla GANs prevent any control over the generation of a specific type of synthetic data.

2.2. Deep Convolutional Generative Adversarial Networks

2.2.1. Concept

Deep Convolutional Generative Adversarial Networks or DCGANs were introduced by (Radford et al., 2016). Owing to the unstable nature of GANs, training a GAN can be highly tricky, with even a small change in architecture or parameters giving nonsensical outputs. Previous attempts to include the benefits of Convolutional Neural Networks (CNN), mainly spatial correlation and parameter sharing to GANs have been unsuccessful for this reason. The original paper on DCGANs tries to bridge the gap between GANs and CNNs through careful model architecture selection and hyperparameter tuning.

2.2.2. Model Architecture

After extensive exploration, the authors were able to identify a specific implementation of CNNs which was considerably different from traditional CNN implementation in supervised learning that worked with GANs.

The first difference was using only convolutions to learn spatial features. This was achieved by using strided convolutions. This eliminates the use of spatial averaging functions like max-pooling or average-pooling.

The second difference was using only convolution layers and completely eliminating the fully connected layer at the end.

The third difference was using Batch Normalization which normalizes batch input to zero mean and unit variance. This helped the gradient flow in the deeper models and prevented mode collapse. Mode collapse is a scenario where the Generator only generates one kind of image. However, simply using Batch Normalization in all layers was also unstable. So, they did not use Batch Normalization layer in Generator output and Discriminator input layer.

The fourth and final difference was using ReLU activation

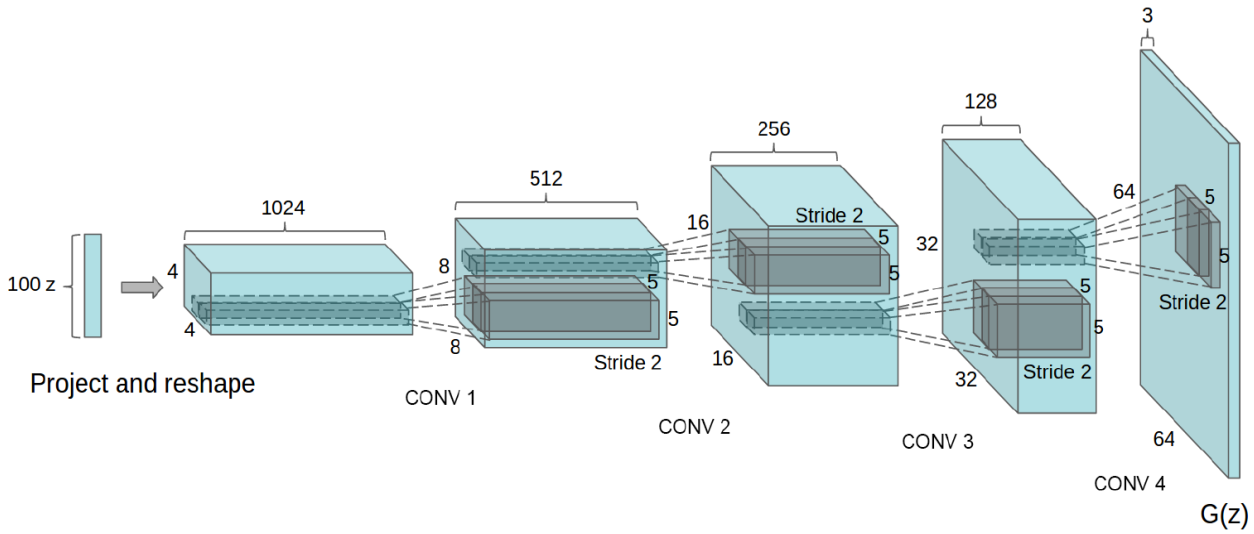

Figure 1. DCGAN Model Architecture. Image taken from (Radford et al., 2016)

in Generator except for the output layer, which used the TanH layer to bring pixel values to the [-1,1] range. For the Discriminator, the authors used Leaky ReLU activation for all layers.

2.2.3. Advantages / Disadvantages

The most prominent advantage of DCGAN was that it was able to utilize the benefits of CNNs in GANs. Spatial correlation allowed DCGANs to produce better-quality images, whereas the parameter sharing reduced the number of learnable parameters and improved the training speed.

However, DCGANs had some major disadvantages which should be brought to light. DCGANs are highly prone to model instability and even the slightest change in model structure or hyperparameters than what was given in the original paper causes the Generator to produce nonsensical output or complete mode collapse. Also, since DCGANs are also meant to be unsupervised, they do not have control over what class of image is generated.

2.3. Wasserstein GANs

2.3.1. Concept

The Wasserstein Generative Adversarial Networks or WGANs were introduced by (Arjovsky et al., 2017). They proposed that using Earth Mover (EM) distance, also called Wasserstein-1, might have better convergence properties than using Jenses-Shannon (JS) divergence. However, the EM distance requires taking an infimum over all joint distributions between the true data distribution and generated data distribution, which is not possible. But, the Kantorovich-Rubinstein duality proposes another version of calculating Wasserstein Distance, which seems tractable.

$$W(P_r, P_\theta) = \sup_{||f||_L \leq 1} E[D(x)] - E[D(G(z)] \quad (4)$$

where $W(P_r, P_\theta)$ is the Wasserstein Distance between P_r and P_z, $x \sim P_r$ and $z \sim P_z$. This involves calculating supremum over all 1-Lipschitz functions D.

Now, the task is to enforce the 1-Lipschitz constraint on the function. In this paper, the 1-Lipschitz constraint was enforced by gradient weight clipping. This procedure of using Wasserstein Loss in GANs was termed Wasserstein GANs.

However, as mentioned by the authors of this paper, gradient weight clipping is a rudimentary way of enforcing the 1-Lipschitz constraint. If the clipping parameter is too large, the weights may take a long time to train till optimality. If the clipping parameter is too small, it may cause vanishing gradients. So, the clipping parameter is another hyperparameter that involves careful tuning.

2.3.2. Algorithm

The algorithm for the WGAN is mentioned in Algorithm 2

Algorithm 2 Wasserstein GANs. The parameters used were Learning Rate $\alpha = 0.00005$, Clipping Parameter $c = 0.01$, Batch Size $M = 64$, No. of iteration of training Critic (Discriminator) $n_{critic} = 5$

Initialize Critic Parameters w_0 and Generator Parameters θ_0
for number of epochs **do**
  for $t = 0, \ldots, n_{critic}$ **do**
    Get minibatch of latent vectors $[z^1 \ldots z^M]$
    Get minibatch of true data $[x^1 \ldots x^M]$
    Get Critic Loss Gradient

$$g_w \leftarrow \nabla_w \frac{1}{M} \sum_{i=1}^{M} [f_w(x^i) - f_w(g_\theta(z^i))] \quad (5)$$

$$w \leftarrow w + \alpha \cdot RMSProp(w, g_w) \quad (6)$$

$$w \leftarrow clip(w, -c, c) \quad (7)$$

  end for
  Get Generator Loss Gradient

$$g_\theta \leftarrow -\nabla_\theta \frac{1}{M} \sum_{i=1}^{M} [f_w(g_\theta(z^i)] \quad (8)$$

$$\theta \leftarrow \theta + \alpha \cdot RMSProp(\theta, g_\theta) \quad (9)$$

end for

2.3.3. Advantages / Disadvantages

The major advantage of WGAN was that it made the training much more stable than what was previously offered by other GAN variants. The authors claim that with WGAN, it is theoretically possible to train the critic to optimality, and when the critic is trained, it can be used to train the generator well. Also, WGANs are much more robust to change in architecture and hyperparameters and can work well with suboptimal choices for them. In addition to that, the authors claim that WGAN also navigates the problem of mode collapse with ease.

The disadvantages of WGANs majorly include the choice of another hyperparameter called the clipping parameter. As explained earlier, the choice of the correct clipping parameter is imperative to optimal performance when using WGANs.

2.4. Wasserstein GANs with Gradient Penalty

2.4.1. Concept

Wasserstein GANs with Gradient Penalty was introduced in (Gulrajani et al., 2017). As mentioned by authors in (Arjovsky et al., 2017), gradient clipping is not an effective way to enforce the 1-Lipschitz constraint. In the paper (Gulrajani et al., 2017), the authors introduce another method called Gradient Penalty to enforce the 1-Lipschitz constraint. They propose penalizing the cost function with the norm of the gradient of the critic with respect to the inputs.

They identified some avenues where using gradient clipping can lead to optimization difficulties. Firstly, gradient clipping acts like a strong regularizer and biases the model towards a much simpler function. This leads to capacity underuse. Secondly, different values of gradient clipping parameters can lead to vanishing or exploding gradients, both of which are detrimental to the optimization problem.

To this extent, they propose gradient penalty as an alternative to gradient clipping. According to the authors, a function is 1-Lipschitz if and only if it has gradient norm equal to 1 almost everywhere. So, they directly penalize the Wasserstein cost function with the Lagrangian of gradient norm being equal to 1.

$$L = E[D(G(z))] - E[D(x)] + GP \quad (10)$$

$$GP = \lambda \cdot E[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \quad (11)$$

Here, $\hat{x}$ is the interpolated image between true image x and synthetic image $G(z)$. The penalty coefficient $\lambda = 10$ was used across the experiment. Also, since the gradient penalty is calculated per-input basis, batch normalization was removed in critic training, because it changes the form of input based on the batch. The authors suggest dropping batch normalization completely or utilizing layer normalization instead.

2.4.2. Algorithm

The algorithm for WGAN with Gradient Penalty is given in Algorithm 3. This algorithm focuses on independently calculating loss for each input in a batch. In my experiment, I used a variant of this algorithm where I use Layer Normalization to enable batch processing.

2.4.3. Advantages / Disadvantages

The major advantages of WGAN with Gradient Penalty are improved image quality and robustness to change in architecture and hyperparameters. WGAN with Gradient Clipping was an improvement over previously used GAN variants, however, it was still prone to encountering problems like mode collapse where the Generator only generates one class of outputs. WGAN with Gradient Penalty was an improvement over this, and provides additional robustness to mode collapse and nonsensical outputs. The authors also claim that using Gradient Penalty may improve the training speed in certain scenarios.

While WGAN with Gradient Penalty provides better stability and image quality than WGAN with Weight Clipping,

it does not come without cost. Gradient Penalty involves the calculation of 2-norm of the gradient with respect to the input. This is computationally heavy and may increase the training time.

Algorithm 3 Wasserstein GANs with Gradient Penalty. The parameters used were Learning Rate $\alpha = 0.0001$, Gradient Lagrangian Multiplier $\lambda = 10$, No. of iteration of training Critic (Discriminator) $n_{critic} = 5$, ADAM First Moment β_1 = 0, ADAM Second Moment $\beta_2 = 0.9$

Initialize Critic Parameters w_0 and Generator Parameters θ_0
for number of epochs **do**
  for $t = 0, \ldots, n_{critic}$ **do**
    for $i = 1, \ldots, M$ **do**
      Get latent vector $z \sim P_z$, true data $x \sim P_x$, $\epsilon \sim U[0, 1]$

$$\tilde{x} \leftarrow G(z) \tag{12}$$

$$\hat{x} \leftarrow \epsilon \cdot x + (1 - \epsilon) \cdot \tilde{x} \tag{13}$$

$$GP \leftarrow \lambda((||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2) \tag{14}$$

$$L^i \leftarrow D(\tilde{x}) - D(x) + GP \tag{15}$$

    end for

$$w \leftarrow ADAM(\frac{1}{M} \nabla_w \sum_{i=1}^{M} L^i) \tag{16}$$

  end for

$$\theta \leftarrow ADAM(\frac{1}{M} \nabla_\theta \sum_{i=1}^{M} -D(G(z))) \tag{17}$$

end for

2.5. Conditional GANs

2.5.1. Concept

Conditional Generative Adversarial Networks or cGANs were introduced in (Mirza & Osindero, 2014). It was a revolutionary paper, which opened avenues to generate synthetic data conditioned on an input. The input is usually a class label for which the data needs to be generated. GANs can be conditioned by including the label as an input to the Generator and Discriminator. In this scenario, the objective function gets changed to:

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[logD(x|y)] + E_{z \sim p_z(z)}[log(1 - D(G(z|y)))] \tag{18}$$

Instead of simply using the true data x, we use the true data given an input label $x|y$. Similarly, to generate an image, instead of simply using the latent variable z, we use the latent variable given an input label $z|y$.

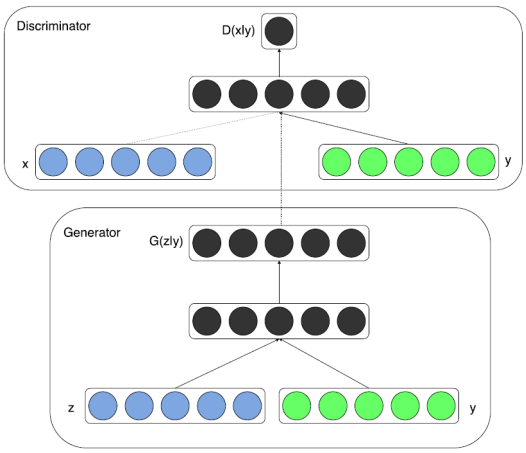

Figure 2. Conditional GAN Architecture. Image taken from (Mirza & Osindero, 2014)

2.5.2. Model Architecture

The model architecture remains relatively the same as the unconditional version. The only difference is the inclusion of an input label in the Generator and Discriminator functions. In the Discriminator, the input label can be embedded in the form of an additional channel of the input image. In the Generator, the input label can be embedded in a vector of arbitrary size and be appended to the latent vector. Figure 2 shows the model architecture of a cGAN.

2.5.3. Advantages / Disadvantages

The paper tackles a very crucial element of synthetic data generation, i.e., synthetic data generation based on user input. This method can be used with any GAN architecture, however, the performance of the final model will depend on the architecture used.

3. Method

I used a combination of the above-defined concepts for the reimplementation project. The basic methodology is shown in Figure 3 and Figure 4 and underlined below:

1. The basic structure of the Critic and Generator was taken from (Radford et al., 2016) with some modifications. In Critic, I used 4x4 convolution blocks with stride=2 to reduce spatial dimensions. Instead of using Batch Normalization, I used Layer Normalization as mentioned in (Gulrajani et al., 2017). In Generator, I used transpose convolution blocks with stride=2 to

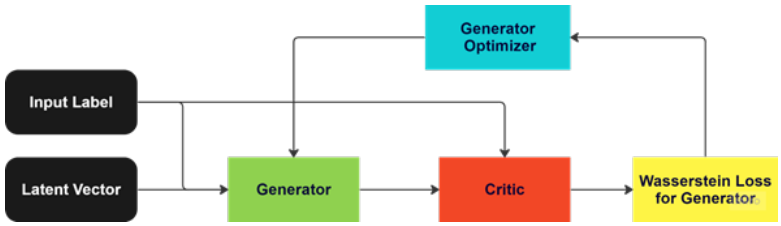

Figure 3. Generator Training

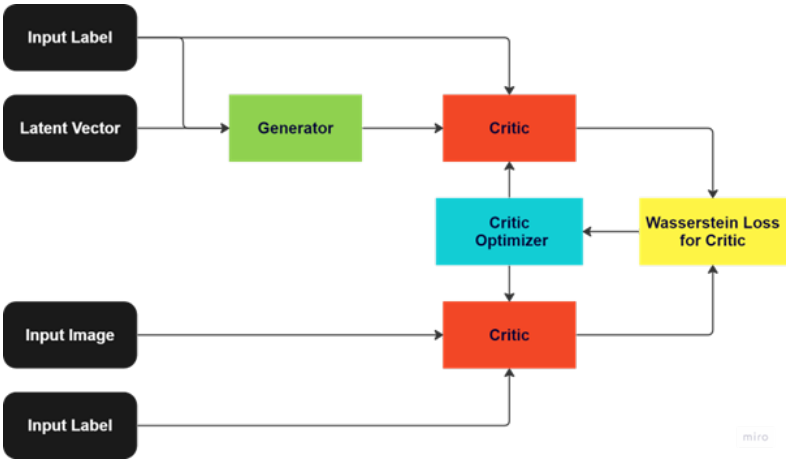

Figure 4. Critic Training

increase spatial dimensions.

2. For the conditional capabilities, I modified the Generator and Discriminator. In Discriminator, I embedded the input label as an additional channel of the image. In Generator, I embedded the input label to a vector of same size as the latent variable. Corresponding modifications were made to the architecture.

3. The objective function used was Wasserstein Loss with Gradient Penalty as described in (Gulrajani et al., 2017) with few modifications. Instead of calculating the loss of each input in a batch, I used Layer Normalization to process one batch at a time. The rationale behind this choice was using independent input images limits the ability to use GPU processing. This would significantly increase the training time. Similarly, Gradient Penalty was calculated on a per-batch basis.

4. The optimizer used was ADAM with hyperparameters as described in (Gulrajani et al., 2017). I explored other hyperparameter choices, but these gave the best results.

5. I also created a structure that would create model and optimizer checkpoints after a certain number of epochs. This would allow us to combat model overfitting due to overtraining and allow finetuning.

4. Experiment

The experiment was performed on the CIFAR-10 dataset resized to (64,64). The following combinations of hyperparameters were tested:

Batch Size	[32, 64]
Learning Rate	[5e-3, 1e-3, 5e-4, 1e-4, 5e-5]
Betas	[(0.9, 0.99), (0.0, 0.9)]

The best combination achieved was Batch Size = 64, Learning Rate = 1e-4, Adam Betas = (0.0, 0.9). This combination was trained for 100 epochs with checkpointing after every 10 epochs. The total training time was approximately 18 hours on Google Colab T4 GPU. The best resulting checkpoint was Checkpoint 4, i.e., after 50 epochs after which the model started overfitting. Figure 5 shows the Generator and Critic loss tally when trained for 50 epochs. Figure 6 shows the resulting generated images conditioned on respective labels.

From the loss tally, we can see that initially the Generator loss increases and later it starts to decrease. I believe this is due to the fact that since we are using Wasserstein Loss and multiple training iterations for Critic for 1 iteration of Generator, the GAN tries to optimize critic before the Generator. As the Critic gets better at distinguishing between real and synthetic images, the generator loss increases.

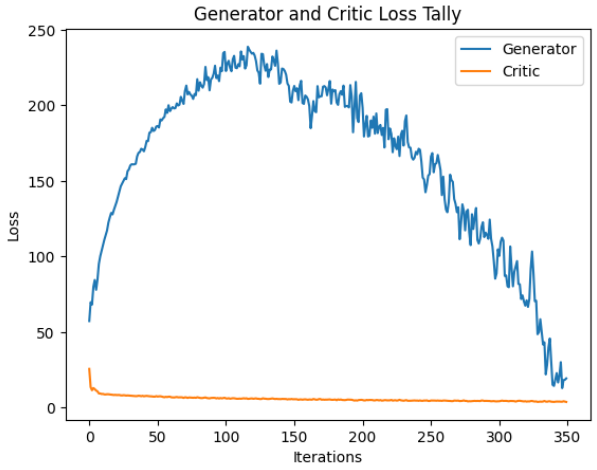

Figure 5. Generator and Critic Loss Tally

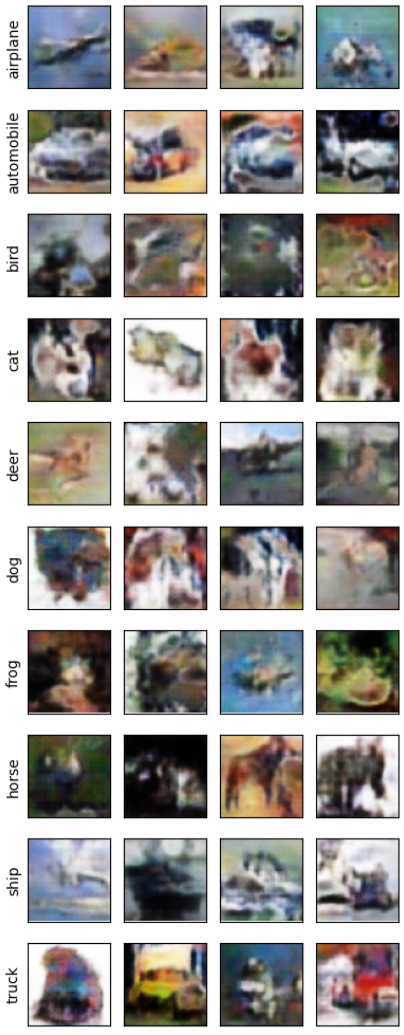

Figure 6. Generated Output

When the Critic reaches convergence, then the Generator loss starts decreasing. Also, from the generated images we can see that the model performed quite well. The model correctly identifies the label and generates a corresponding image. However, the quality of the generated image can be improved by using more complex architectures like ResNet.

5. Conclusion

This paper presents a successful reimplementation of the original paper "Conditional Generative Adversarial Networks" by (Mirza & Osindero, 2014). The model was successfully able to generate synthetic images conditioned on a class label. However, the image quality can be further improved by using more complex architectures and careful hyperparameter tuning. A similar experiment done in "Improved Training of Wasserstein GANs" by (Gulrajani et al., 2017) resulted in higher-quality synthetic images. The authors used a ResNet architecture with AC-GAN Conditioning for the high-quality generated images.

Figure 7. ResNet Baseline Generated Images. Image taken from (Gulrajani et al., 2017)

6. GitHub Repository

Please refer to the GitHub repository at:
`https://github.com/suhaibk24/ECE50024---Project---Conditional-GANs`

7. Acknowledgment

I am grateful to Professor Guo for providing unwavering support throughout the reimplementation process. His guid-

ance and lectures were instrumental in equipping me with the necessary knowledge and confidence to successfully re-implement this project. I would also like to express my gratitude to the Teaching Assistants for helping me navigate several doubts and concerns.

References

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein gan, 2017.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks, 2014.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. Improved training of wasserstein gans, 2017.

Mirza, M. and Osindero, S. Conditional generative adversarial nets, 2014.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.